GeDi: Solving multiobjective linear programs via the generalized dichotomic search.

Nathan Adelgren
Department of Mathematics and Computer Science
Edinboro University
Edinboro, PA 16444
nadelgren@edinboro.edu

Daniel Bennett
Department of Mathematics and Computer Science
Edinboro University
Edinboro, PA 16444
dbennett@edinboro.edu

ABSTRACT

We present GeDi, a new software package which computes all Pareto solutions for a multiobjective linear program by utilizing a version of the dichotomic search [3, 15] which has been generalized for any finite number of objectives. GeDi is employed to solve a substantial set of problems, including those available through MOPLIB [32]. The performance of GeDi is compared against that of BenSolve [33, 34], which is based on Benson's algorithm and its extensions [7, 9, 10], Inner [16, 17], and PolySCIP [12, 13] using a variety of metrics. Results indicate that GeDi is competitive with current state of the art software.

KEYWORDS

Multiobjective, Pareto, Objective-space, Dichotomic search

1. Introduction

This paper introduces GeDi, a new software package designed to generate the entire Pareto set of a multiobjective linear program (MOLP), which is formulated as

$$\min_x \left\{ \begin{array}{c} f_1(x) := {c^1}^\intercal x \\ \vdots \\ f_p(x) := {c^p}^\intercal x \end{array} \right\} \quad \text{s.t.} \quad x \in X \tag{1}$$

where $X := \{x \in \mathbb{R}^n \colon Ax \leq b,\ l_i \leq x_i \leq u_i\ \forall i\}$. Note that for the above model we permit $l_i = -\infty$ or $u_i = \infty$ for any i, but we do make the assumption that the objectives $f_1, \ldots, f_p$ are conflicting, i.e., for distinct $i, j \in \{1, \ldots, p\}$ there does not exist $x \in X$ that minimizes both f_i and f_j simultaneously.

Multiobjective optimization problems have been studied for decades; see [22, 23] for relevant surveys. The earliest algorithms developed for solving MOLP were so-called variable space (or constraint space) search algorithms which sought to compute all solutions by working in the space X directly [4, 5, 20, 21, 25, 29, 40]. Later, many researchers determined that a representative subset of the solutions to MOLP could be obtained by working in the space

$$Y := \{y \in \mathbb{R}^p : y = (f_1(x), \ldots, f_p(x)), x \in X\}, \tag{2}$$

often with far less computational effort. Thus, so-called objective space (or criterion space) search algorithms were born [10, 17–19, 24, 27, 31]. An alternative approach to solving MOLP is to apply to MOLP a well known scalarization technique known as the weighted-sum method (described in more detail later) and then work in the so-called weight space to compute the same representative subset of solutions that can be achieved via an objective space search [8, 13, 36]. The algorithms we present in this work and which are employed by GeDi utilize an objective space search strategy.

2. Definitions and Notation

The idea of optimality for single objective optimization is replaced with the idea of *efficiency* in multiobjective problems. Consider MOLP (1). For each $x \in X$, define $y(x) \in Y$ so that $y(x) := (f_1(x), \ldots, f_p(x))$. Given distinct $x, x' \in X$, we say that $y(x)$ *dominates* $y(x')$ if $y(x) \leq y(x')$, or equivalently, if $y(x')$ is contained in the set $y(x) + \mathbb{R}^p_{\geq 0}$. We denote this relationship as $y \succ y'$. We then say that $x \in X$ is *efficient* if there is no $x' \in X$ such that $y(x') \succ y(x)$. The set of efficient solutions is denoted by X_E. Additionally, $y(x) \in Y$ is called *Pareto optimal* or *nondominated* if $x \in X_E$.

As mentioned in Section 1., there are many strategies that are employed when solving MOLP (1). However, these strategies can all be categorized into one of two groups based on whether they seek to compute: (i) each $x \in X_E$ or, (ii) a set $X' \subseteq X_E$ such that for each $y \in Y_N$, there exists at least one $x \in X'$ such that $y = y(x)$. The procedures proposed in this paper are of the latter type.

3. The Generalized Dichotomic Search

In this section we discuss the primary algorithm employed by GeDi, a generalization of the dichotomic search proposed by Aneja and Nair [3] for solving MOLP when $p = 2$. Our proposed generalization is designed to solve MOLP for any finite p, and henceforth we refer to it as the generalized dichotomic search (GDS).

GDS is an iterative process which maintains and updates a set $\mathcal{S}$ of $(p-1)$-dimensional simplices such that at every iteration, $\mathcal{S} + \mathbb{R}^p_{\geq 0} \subseteq Y_N + \mathbb{R}^p_{\geq 0}$ and at termination, $\mathcal{S} + \mathbb{R}^p_{\geq 0} = Y_N + \mathbb{R}^p_{\geq 0}$. Given a $(p-1)$-dimensional simplex $s \in \mathcal{S}$, let $y^1_s, \ldots, y^p_s$ denote its p extreme points and let n_s denote the normal vector of the $(p-1)$-dimensional hyperplane aff(s), where aff$(\cdot)$ denotes the affine hull.

I left off here.

GDS relies heavily on the well known *weighted sum problem* (WSP), which utilizes a weight vector $\lambda \in \mathbb{R}^p$ to scalarize MOLP. WSP is formulated as

$$WSP(\lambda) := \min_x \left\{ \sum_{i=1}^{p} \lambda_i f_i(x) \right\} \quad \text{s.t.} \quad x \in X. \tag{3}$$

Throughout this work we exploit the properties of *slice problems* – continuous multiobjective programs obtained by fixing the integer variables to some feasible values. Let us decompose each $x \in X_I$ as $(x^{\mathbb{Z}}, x^{\mathbb{R}})$, where $x^{\mathbb{Z}}$ and $x^{\mathbb{R}}$ denote the integer and continuous subvectors of x, respectively.

The slice problem for a given $x^* \in X_I$ is

$$\mathbb{P}(x^*) := \max\left\{\boldsymbol{f}(x) : x^{\mathbb{Z}} = (x^*)^{\mathbb{Z}},\, x \in X_I\right\}$$

and we use $\mathcal{P}(x^*)$ to denote the set of Pareto optimal solutions to $\mathbb{P}(x^*)$. We have $Y_N = \mathcal{ND}(\cup_{x\in X_I}\mathcal{P}(x))$. Hence, in this work we seek to find Y_N by constructing $\mathcal{ND}(\cup_{x\in X_I}\mathcal{P}(x))$.

The idea behind the slice problem exploitation method (SPEM) is, in general, the same as all other objective (or criterion) space search algorithms – to iteratively break $\mathcal{OS}$ into subregions and solve single objective MILPs which are carefully formulated so as to ensure that all Pareto solutions within each subregion are computed. In this work, each subregion we consider is rectangular, and we refer to them as *boxes*. We identify a box $b \subset \mathbb{R}^2$ using its northwest and southeast corners (singletons) which we denote $b^{nw} = (b_1^{nw}, b_2^{nw})$ and $b^{se} = (b_1^{se}, b_2^{se})$, respectively. We will also use the notation $b = (b^{nw}, b^{se})$.

SPEM depends on two underlying tools: (i) a dynamic data structure $\mathscr{D}$ capable of taking subsets of Y_I as input and storing only the nondominated subset of all input, and (ii) a procedure for computing $\mathcal{P}(x)$ for a given $x \in X_I$. For the former, we utilize the data structure of Adelgren et al. [2]. The latter can be accomplished using the the dichotomic search [3, 15] or parametric simplex [22, 39] algorithms, though we utilize a different procedure which is described later in this section (Algorithm 3). For a given $x \in X_I$, $\mathcal{P}(x)$ is a subset of $\mathbb{R}^2$ consisting of either a singleton or a piecewise-linear, monotonically decreasing curve. Hence, Y_N is formed as a union of singletons and line segments with negative slopes in $\mathbb{R}^2$. Consequently, the data structure $\mathscr{D}$ is designed specifically to store either singletons or line segments with negative slopes. We will use $z \in \mathscr{D}$ to indicate a singleton or segment that is stored in $\mathscr{D}$. Furthermore, each $z \in \mathscr{D}$ is identified by its northwest and southeast extreme points, which we denote $z^{nw} = (z_1^{nw}, z_2^{nw})$ and $z^{se} = (z_1^{se}, z_2^{se})$, respectively. Note that if $z \in \mathscr{D}$ is a singleton, we have $z^{nw} = z^{se}$. The process of adding an additional singleton or line segment (or a union of several singletons and line segments) to $\mathscr{D}$ consists of: (i) removing from storage the subset of any $z \in \mathscr{D}$ which is dominated by the input, (ii) adding to storage the subset of the input which is not dominated by any currently stored $z \in \mathscr{D}$. We denote this procedure as $\textsc{Insert}(\cdot, \mathscr{D})$.

The basic strategy of SPEM is very simple and the pseudocode is provided in Algorithm 1. Through use of a "for" loop, on line 4 we solve two single objective MILPs, one for each $f_1(x)$ and

Algorithm 1 Slice Problem Exploitation Method.
Input: Instance $\mathcal{I}$ of BOMILP.
Output: The Pareto set of $\mathcal{I}$, stored in $\mathscr{D}$.

```
1: function SPEM($\mathcal{I}$)
2:     Let $\mathscr{D} = \emptyset$.
3:     for $k \in \{1, 2\}$ do
4:         Let $x_I^k = \text{argmax}\{f_k(x) : x \in X_I\}$.
5:         $\mathcal{P}(x_I^k) = \textsc{GeneratePareto}(x_I^k)$.
6:         $\textsc{Insert}(\mathcal{P}(x_I^k), \mathscr{D})$.
7:     Let $b^1 = (\boldsymbol{f}(x_I^2), \boldsymbol{f}(x_I^1))$ and $b^2 = \emptyset$.
8:     $\mathscr{D} = \textsc{ExploreBox}(b^1, b^2, \mathscr{D})$.
9:     Return $\mathscr{D}$.
```

$f_2(x)$. The image of the solution of the MILP associated with $f_2(x)$ then forms the northwest corner of the first box we will explore. Similarly, the image of the solution of the MILP associated with

$f_1(x)$ forms the southeast corner. For use in later algorithms, we define the point

$$z^* = \left(f_1(x_I^1), f_2(x_I^2)\right).$$

On line 8 we call the function ExploreBox, which serves several purposes: (i) the set Y_N^1 of Pareto solutions to BOMILP (**??**) which lie within b^1 (the first argument) is computed, (ii) Y_N^1 is added to $\mathscr{D}$, (iii) under a special set of circumstances the orientation of b^2 (the second argument) is modified, and (iv) $\mathscr{D}$ is returned. Clearly, the key aspect of SPEM is this ExploreBox procedure. We provide the pseudocode for ExploreBox in Algorithm 2. Note that z^1, defined on line 2, is the

Algorithm 2 Explore a rectangular subset of $\mathcal{OS}$.
Input: Two rectangular subsets of $\mathcal{OS}$, denoted b^1 and b^2, and data structure $\mathscr{D}$.
Output: $\mathscr{D}$ (potentially updated).

```
1: function ExploreBox($b^1, b^2, \mathscr{D}$)
2:     Let $z^1 := \{z \in \mathscr{D} : z \cap b^1 = z$ and $z^{nw} \leq \hat{z}^{nw}$ for all $\hat{z} \in \mathscr{D} \setminus z$ such that $\hat{z} \cap b^1 = \hat{z}\}$.
3:     Let $z^2 := \{z \in \mathscr{D} \setminus z^1 : z^{nw} \geq z^{1,se}$ and $z^{nw} \leq \hat{z}^{nw}$ for all $\hat{z} \in \mathscr{D} \setminus z^1$ such that $\hat{z}^{nw} \geq z^{1,se}\}$.
4:     if $z^{1,se} = z^{2,nw}$ then
5:         if $z^2 \cap b^1 \neq z^{2,nw}$ then
6:             Define $b^3$ so that $b^{3,nw} = b^{1,nw}$ and $b^{3,se} = (b_1^{1,se}, z_2^{1,se})$.
7:             Define $b^4$ so that $b^{4,nw} = z^{1,se}$ and $b^{4,se} = b^{1,se}$.
8:             $\mathscr{D}$ = ExploreBox($b^3, b^4, \mathscr{D}$).
9:             $\mathscr{D}$ = ExploreBox($b^4, \emptyset, \mathscr{D}$).
10:        else
11:            Let $s$ denote the slope of $z^1$ and let $x_I^w = \text{argmax}\{f_w(x) := f_1(x) - \frac{1}{s} f_2(x) : b_1^{1,nw} \leq f_1(x) \leq b_1^{1,se}, b_2^{1,se} \leq f_2(x) \leq b_2^{1,nw}, x \in X_I\}$.
12:            $\mathcal{P}(x_I^w)$ = GeneratePareto($x_I^w$).
13:            Insert($\mathcal{P}(x_I^w), \mathscr{D}$).
14:            if $f_w(x_I^w) = z_1^{1,nw} - \frac{1}{s} z_2^{1,nw}$ then return $\mathscr{D}$.
15:            else
16:                if $b^2 \neq \emptyset$ then
17:                    Find $z \in \mathscr{D}$ such that $f_w(x_I^w) \cap z \neq \emptyset$.
18:                    Modify $b^1$ so that $b_2^{1,se} = z_2^{nw}$.
19:                    Modify $b^2$ so that $b^{2,nw} = z^{nw}$.
20:                $\mathscr{D}$ = ExploreBox($b^1, \emptyset, \mathscr{D}$).
21:    else
22:        Let $\beta = \frac{z_2^* - z_2^{2,nw}}{z_1^* - z_1^{1,se} + z_2^* - z_2^{2,nw}}$.
23:        Let $(t^c, x_I^c) = \text{argmin}\{t : t \geq \beta(z_1^* - f_1(x)), t \geq (1-\beta)(z_2^* - f_2(x)), t \in \mathbb{R}, x \in X_I\}$.
24:        $\mathcal{P}(x_I^c)$ = GeneratePareto($x_I^c$).
25:        Insert($\mathcal{P}(x_I^c), \mathscr{D}$).
26:        if $t^c = \beta(z_1^* - z_1^{1,se})$ then
27:            Define $b^3$ so that $b^{3,nw} = b^{1,nw}$ and $b^{3,se} = z^{1,se}$.
28:            Define $b^4$ so that $b^{4,nw} = z^{2,nw}$ and $b^{4,se} = b^{1,se}$.
29:            $\mathscr{D}$ = ExploreBox($b^3, \emptyset, \mathscr{D}$).
30:            $\mathscr{D}$ = ExploreBox($b^4, \emptyset, \mathscr{D}$).
31:        else $\mathscr{D}$ = ExploreBox($b^1, \emptyset, \mathscr{D}$).
32:    Return $\mathscr{D}$.
```

left-most $z \in \mathscr{D}$ such that z is contained in b^1. Similarly, z^2, defined on line 3, is the left-most

$z \in \mathscr{D}$ that lies to the right of z^1. Consider the following two cases:

Case 1: There is no separation between z^1 and z^2 (i.e. the "if" statement on line 4 is satisfied).

Case 2: There is separation between z^1 and z^2.

Case 1 is processed on lines 4–20. If b^1 contains more of z^2 than $z^{2,nw}$ we use the horizontal line containing $z^{1,se}$ to split b^1 into two sub-boxes, and then these sub-boxes are explored, beginning with the upper box. If $b^1 \cap z^2 = z^{2,nw}$ we use the weighted sum scalarization technique [26] to determine whether or not any Pareto solutions exist above and to the right of z^1. If such solutions are discovered (i.e., the "if" statement on line 14 is not satisfied), we re-explore b^1. Otherwise, we know $z^1 \in Y_N$. (Note that we will clarify the use of lines 16–19 via the example presented in the next section.)

Case 2 is processed on lines 21–31. We use the weighted Chebyshev scalarization technique [35, 38] to determine whether or not any Pareto solutions exist to the right of z^1 and above z^2. If such solutions are discovered (i.e., the "if" statement on line 26 is not satisfied), we re-explore b^1. Otherwise, we know that if b^1 contains any Pareto solutions, they lie within either $b^3 = (b^{1,nw}, z^{1,se})$ or $b^4 = (z^{2,nw}, b^{1,se})$. Hence, we explore each of these boxes, beginning with b^3.

The final algorithm we present in this section is that of the GeneratePareto procedure that we use for generating $\mathcal{P}(x^*)$ for a given $x^* \in X_I$. For use in this algorithm, we introduce the following LP.

$$\mathscr{P}(\alpha, x^*) := \max\{f_1(x) + \alpha f_2(x) : x^{\mathbb{Z}} = (x^*)^{\mathbb{Z}}, x \in X_I\} \quad (4)$$

The pseudocode for GeneratePareto is given in Algorithm 3. We note that this algorithm

Algorithm 3 Generate $\mathcal{P}(x)$
Input: $x^* \in X_I$.
Output: $\mathcal{P}(x^*)$.

```
1: function GeneratePareto($x$)
2:     Set $\mathcal{B} = \emptyset$.
3:     Let $y' = \boldsymbol{f}(x')$, where $x' = \text{argmax}\{f_2(x) : x^{\mathbb{Z}} = (x^*)^{\mathbb{Z}}, x \in X_I\}$.
4:     Let $y = \boldsymbol{f}(x)$, where $x$ is the optimal solution to $\mathscr{P}(0, x^*)$.
5:     while $y' \neq y$ do
6:         Use sensitivity analysis to obtain an interval $[\alpha', \alpha'']$ such that $x$ is optimal to $\mathscr{P}(\alpha, x^*)$
       for all $\alpha \in [\alpha', \alpha'']$.
7:         Let $\alpha^*$ be the negative reciprocal of the slope of the line segment connecting $y$ and $y'$.
8:         Update $x$ to be the optimal solution of $\mathscr{P}(\alpha''+\epsilon, x^*)$ for sufficiently small $\epsilon \in (0, \alpha^* - \alpha'']$.
9:         if $\boldsymbol{f}(x) \neq y$ then
10:            Add the line segment connecting $\boldsymbol{f}(x)$ and $y$ to $\mathcal{B}$. Update $y$ to be $\boldsymbol{f}(x)$.
11:    Return $\mathcal{B}$.
```

depends on a tolerance level ϵ and, in general, produces an approximation of $\mathcal{P}(x)$ because if in any iteration, ϵ is not chosen sufficiently small, GeneratePareto can fail to compute an extreme point of $\mathcal{P}(x)$. We also note, however, that this does not affect the correctness of SPEM because if GeneratePareto fails to compute any extreme point of $\mathcal{P}(x)$, this extreme point will later be discovered as the solution to a MILP during a call to ExploreBox. $\mathcal{P}(x)$ can then be updated appropriately when this happens. Additionally, we point out that with ϵ set to 10^{-7} during computational testing, GeneratePareto correctly computed all extreme points for the Pareto sets of each slice problem considered for each instance of BOMILP solved in Section 6..

The finiteness and correctness of SPEM are argued in Section 5.. We now proceed to Section 4., where we present an example of using SPEM to solve a small instance of BOMILP.

4. An Illustrative Example

Consider an instance $\mathcal{I}$ of BOMILP with five distinct feasible $x \in X_I$. Suppose the Pareto sets of the slice problems associated with each of these solutions are as displayed in Figure 1a. We

(a) Pareto sets of all slice problems.

(b) $\mathcal{D}$ and b^1 after lines 2–7 of SPEM($\mathcal{I}$).

Figure 1

process instance $\mathcal{I}$ by calling SPEM($\mathcal{I}$). The solutions x_I^1 and x_I^2, as computed on line 4 of Algorithm 1, correspond to x^4 and x^1 in Figure 1a. Additionally, $\boldsymbol{f}(x^4)$ and $\boldsymbol{f}(x^1)$ are depicted as a star and square, respectively, in Figure 1a. We now create $b^1 = (\boldsymbol{f}(x^1), \boldsymbol{f}(x^4))$ and call ExploreBox($b^1, \emptyset, \mathscr{D}$). The segments currently stored in $\mathscr{D}$ and b^1 (outlined by a dashed line) are displayed in Figure 1b. Segments z^1 and z^2, computed on lines 2 and 3 of Algorithm 2, are also depicted in Figure 1b. Since $z^{1,se} = z^{2,nw}$ and $z^2 \cap b^1 = z^2$, we split b^1 into b^3 and b^4, where $b^{3,nw} = b^{1,nw}$, $b^{3,se} = (b_1^{1,se}, z_2^{1,se})$, $b^{4,nw} = z^{1,se}$ and $b^{4,se} = b^{1,se}$. Figure 2a shows b^3 (dashed) and b^4 (solid). We now call ExploreBox($b^3, b^4, \mathscr{D}$). In this iteration we reach line 10 of Algorithm 2.

(a) First split of a box, z^1 and z^2 when processing the second box.

(b) New Pareto solution from solving weighted Chebyshev MILP.

Figure 2

By construction, the weighted sum MILP solved on line 11 will reveal a Pareto solution above and to the right of the segment contained in the dashed box, if such a solution exists. Since there are no such solutions, this iteration terminates and we now process box b^4. The segments computed as z^1 and z^2 during this iteration are displayed in Figure 2a. In this situation we reach line 22 of Algorithm 2. By construction, the weighted-Chebyshev MILP solved on line 23 will reveal a Pareto solution to the right of z^1 and to above z^2, if such a solution exists. In this case, a Pareto solution is discovered (depicted as a star in Figure 2b) and x^3 (from Figure 1a) is revealed. As a result, $\mathcal{P}(x^3)$ is computed and added to $\mathscr{D}$ and then we re-process box b^4. The solutions now stored in $\mathscr{D}$, as well as updated segments z^1 and z^2, are shown in Figure 2b. This iteration is analogous to the last, with the exception that no new Pareto solutions are discovered when solving the MILP on line 23. Hence, we reach line 27 of Algorithm 2 and split the current box into two sub-boxes as displayed in Figure 3a. Processing the dashed box of Figure 3a will result in no new Pareto solutions, so we move to the solid box. Initially, processing this box will be analogous to our first two iterations. In fact, no new Pareto solutions will be revealed until we process the dashed box of Figure 3b. The newly discovered solution is shown as a star in Figure 4. We point out that the discovery of this solution is also precisely the type of situation in which we reach lines 18 and 19 of Algorithm

(a) Box split after weighted Chebyshev reveals no new solutions.

(b) Box split for which the next new Pareto solution is discovered.

Figure 3

Figure 4: New Pareto solution found while solving weighted sum MILP, reoriented boxes, and final set of solutions stored in $\mathscr{D}$.

2. Thus, the dashed and solid boxes depicted in Figure 3b are reoriented and become the dashed and solid boxed shown in Figure 4, respectively. The remainder of the iterations required for this example are analogous to those already reviewed, so we cease our discussion here. Note that the final set of Pareto solutions stored in $\mathscr{D}$ appears in Figure 4.

5. Theoretical Results

In this section we argue the correctness and finiteness of SPEM, beginning with the former. Consider the following propositions.

Proposition 1. *Suppose a box $b \subset \mathbb{R}^2$ is being explored using Algorithm 2. If x_I^w is an optimal solution of the weighted sum MILP*

$$\max\{f_w(x) := f_1(x) - \frac{1}{s} f_2(x) : \quad b_1^{1,nw} \le f_1(x) \le b_1^{1,se}, \quad b_2^{1,se} \le f_2(x) \le b_2^{1,nw}, x \in X_I\}$$

solved on line 11, then $\boldsymbol{f}(x_I^w) \in Y_N$. Furthermore, if $f_w(x_I^w) = z_1^{1,nw} - \frac{1}{s} z_2^{1,nw}$ then all $y \in z^1$ are in Y_N.

Proof. We begin by proving the first statement. It is well known [22] that when the feasible set of a weighted sum MILP is X_I and the weights are positive constants, the optimal solution is efficient and thus its image is Pareto. The MILP we solve is constructed with positive weights, but its feasible set is a subset of X_I. Even so, the order in which boxes are explored in SPEM ensures that $\boldsymbol{f}(x_I^w) \in Y_N$ because all boxes that lie above b are explored before b. Thus, if there were to exist $y \in Y_N$ such that $y \succ \boldsymbol{f}(x_I^w)$, y would be discovered before exploring b. Additionally, SPEM is designed so that if a new solution $\hat{y}$ is discovered while processing a box $\hat{b}$, no box processed after $\hat{b}$ will contain points dominated by $\hat{y}$. Hence, the existence of y contradicts the discovery of x_I^w as an optimal solution of the weighted sum MILP.

The proof of the second statement is due to the construction of f_w. Using 1 as the weight on f_1 and the negative reciprocal of the slope of z^1 as the weight on f_2 when constructing f_w ensures that the value of $f_w(x)$ is invariant for any $x \in X_I$ that maps to a point in $\mathcal{OS}$ that lies on z_1. Thus, since $f_w(x_I^w) = z_1^{1,nw} - \frac{1}{s} z_2^{1,nw}$ implies that the value of f_w evaluated at x_I^w and the value of f_w evaluated at the pre-image of $z^{1,nw}$ are equal, the result holds. $\square$

Proposition 2. *Suppose a box $b \subset \mathbb{R}^2$ is being explored using Algorithm 2. If (t^c, x_I^c) is an optimal solution of the weighted Chebyshev MILP*

$$\min\{t : t \geq \beta(z_1^* - f_1(x)),$$
$$t \geq (1-\beta)(z_2^* - f_2(x)), t \in \mathbb{R}, x \in X_I\}$$

solved on line 23, then $\boldsymbol{f}(x_I^c) \in Y_N$.

Proof. This result is already well known [14]. □

Proposition 3. *At termination of SPEM, $y \in Y_N$ if and only if $y \in z$ for some $z \in \mathscr{D}$.*

Proof. ($\Rightarrow$) Consider the first box b explored by SPEM. Initially $\mathscr{D}$ contains at least two z, say z^1 and z^2, such that $z_2^{1,nw} = b_2^{nw}$ and $z_1^{2,se} = b_1^{se}$, i.e., there is not a vertical gap between the top of b and the north-west point of z^1 and there is not a horizontal gap between the right of b and the south-east point of z^2. Then SPEM proceeds by using weighted sum MILPs to find all solutions above and to the right of any $z \in \mathscr{D}$ and weighted Chebyshev MILPs to find all solutions to the right of $\hat{z}$ and above $\tilde{z}$ for any pair of non-intersecting, adjacent $\hat{z}, \tilde{z} \in \mathscr{D}$. If any new solutions are found when solving these MILPs, the process repeats. Hence, at termination there cannot exist any solutions to BOMILP above and to the right of any $z \in \mathscr{D}$ or to the right of $\hat{z}$ and above $\tilde{z}$ for any pair of non-intersecting, adjacent $\hat{z}, \tilde{z} \in \mathscr{D}$. Thus, all $y \in Y_N$ are stored in $\mathscr{D}$.

($\Leftarrow$) This direction of the proof follows directly from Propositions 1 and 2 and the fact that $\mathscr{D}$ is designed so that there cannot exist $z^1, z^2 \in \mathscr{D}$ for which some $y^1 \in z^1$ dominates some $y^2 \in z^2$. □

Proposition 3 ensures the correctness of SPEM. We now establish the finiteness. Consider the following results.

Proposition 4. *SPEM always terminates in finite time.*

Proof. Recall that we assume that BOMILP (**??**) is feasible. Thus solutions will be discovered during execution of Algorithm 1, ensuring that Algorithm 2 is called. When exploring a box b in Algorithm 2 there are only three possibilities: (i) it is shown that no new Pareto solutions exist in b and b is no longer considered, (ii) a slice problem is discovered having a Pareto set which contains at least one point that is within b and Pareto for BOMILP; b is then reprocessed, or (iii) it is shown that a subset of b cannot contain Pareto solutions and two specific sub-boxes are then explored. Of these three possibilities, the finiteness of SPEM comes into question only from (ii). However, we note that by construction, each slice problem having a Pareto set which contains at least one point that is Pareto for BOMILP can be discovered at most once during the execution of SPEM. Moreover, since we assume that neither f_1 nor f_2 are unbounded in X_I, there are a finite number of such slice problems. Thus, SPEM must terminate in a finite number of iterations. □

Proposition 5. *Let n denote the number of $x \in x_I$ having distinct $\mathcal{P}(x)$ for which $\mathcal{P}(x) \cap Y_N \neq \emptyset$, let s_1 and s_2 denote the number of individual line segments and singletons in $\mathbb{R}^2$ that make up Y_N, respectively, and let g denote the number of pairs of adjacent, non-intersecting segments/singletons in Y_N. Then, under the assumption that* GeneratePareto *correctly computes $\mathcal{P}(x^*)$ for a given $x^* \in X_I$, the number of MILPs solved during SPEM is at most $n + s_1 + g + 2$.*

Proof. In the worst case, one MILP solve is needed to reveal each slice problem having a Pareto set which contains a point that is Pareto for BOMILP (this result can be improved by using a heuristic to generate a set of solutions prior to or during the execution of SPEM). This requires

n MILP solves. Recall that at termination of SPEM, Y_N is stored in $\mathscr{D}$. For each z in $\mathscr{D}$ that represents a segment, a weighted sum MILP is solved to prove it is Pareto. This requires solving s_1 MILPs. Additionally, for each pair (z^1, z^2) of adjacent, non-intersecting segments/singletons with $z_1^{1,se} \leq z_1^{2,nw}$ in $\mathscr{D}$, a weighted Chebyshev MILP is solved to prove that no Pareto solutions exist above $z^{2,nw}$ and right of $z^{1,se}$. This requires solving g MILPs. Finally, it is possible that either one (or both) of the two MILPs solved in Algorithm 1 may yield solutions that are dominated by solutions discovered in Algorithm 2. The result follows. □

6. Computational Analysis

We implemented SPEM using the C programming language and the ILOG CPLEX optimization package [28]. We compare the performance of SPEM to two other objective space search algorithms and one variable space search algorithm. The objective space search algorithms we compare to are the triangle splitting algorithm (TSA) of Boland et al. [11] and the ϵ,Tabu–constraint algorithm (ϵT) of Soylu and Yıldız [37]. The variable space algorithm we use is the branch-and-bound technique (BB) of Adelgren and Gupte [1]. All testing was conducted using a personal computer with a 3.40 GHz processor and 16GB of RAM, running Linux Mint 18.

For testing, we utilized instances made available by the authors of [1, 6, 11]. The instances from [6] contained either 20 variables and 20 constraints, 40 variables and 40 constraints, 60 variables and 60 constraints, or 80 variables and 80 constraints. We label these instance sets "Be20," "Be40," "Be60," and "Be80," respectively. In a similar fashion, we label the instances from [11] as "Bo20," "Bo40," "Bo80," "Bo160," and "Bo320." The instances from [1] are generated from classically challenging instances of MILP found in the MIPlib 2010 library [30] by adding a variety of second objectives. As such, we use the same labelling for these instances as was used in [1]. Note that due to the size and difficulty of the majority of these instances, we were only able to utilize a small subset of them in this work.

We compare the performance of all algorithms using two metrics: (i) CPU time, and (ii) number of MILPs solved. The use of CPU time is necessary particularly due to the fact that number of MILPs solved is not a useful metric when comparing to variable space algorithms such as the BB of Adelgren and Gupte [1]. However, CPU time can be somewhat misleading when comparing two objective space search algorithms if the algorithms utilize different underlying MILP solvers, utilize different settings within a given MILP solver, utilize different tolerance measurements for optimality, etc. Thus, when comparing objective space search methods, number of MILPs solved may be a better metric, though it too has its faults since the structure and size of the MILPs solved in one objective space search method can vary greatly from the structure and size of the MILPs solved in another.

We note here that since the code for the ϵT method was not available to us, we cannot report CPU times for this method, but only the number of MILPs solved as reported in [37]. For this same reason, we were unable to utilize the ϵT method on the MIPlib instances from [1]. Additionally, not all instances from [6] were considered in [37], so the numbers of MILPs we report for the ϵT method on these instances are approximated based on the values reported in [37].

The results of our tests are summarized in Tables 1 and 2. In these tables, the symbol "CPU" indices CPU time in seconds and "#M" indicates the number of MILPs solved. From Table 1, recognize that SPEM outperforms TSA in terms of both CPU time and number of MILPs solved for all instance types except Bo160 and Bo320. Additionally, for all instance types, the number of MILPs solved by SPEM is comparable to the number of MILPs solved by ϵT as is reported in [37]. In order to determine the cause of the severe difference in performance between SPEM

Table 1: Results for instances from [6] and [11].

	TSA		ϵT	SPEM		BB
Prob	CPU	#M	#M	CPU	#M	CPU
Be20	0.4	107.4	44.9	0.1	44.0	0.1
Be40	2.4	192.3	75.0	1.0	83.5	0.6
Be60	6.0	213.0	†	3.1	96.9	2.0
Be80	13.5	250.4	128.5	7.2	117.6	4.7
Bo20	0.3	133.2	56.8	0.1	53.6	0.1
Bo40	2.2	428.6	201.2	0.9	205.2	0.2
Bo80	29.6	1609.6	940.6	15.4	964.0	3.4
Bo160	265.6	2969.2	3676.2	319.4	3825.2	71.01
Bo320	2819.2	4399.0	14706.0	8475.4	15561.0	1805.8

Values are averaged over all available instances of each type.
† – Not reported in [37].

Table 2: Results for instances from [1].

	TSA		SPEM		BB
Prob	CPU	#M	CPU	#M	CPU
bienst2 – o	829.5	43	722.2	20	564.4
binkar10_1 – d	831.0	58	1355.9	45	2419.6
neos13 – c	366.7	14	110.3	6	153.2
neos-1396125 – a	198.5	4	48.8	3	413.5
neos-693347 – c	216.5	7	122.2	4	168.7
neos-693347 – d	233.6	7	122.1	4	178.7
neos-916792 – a	148.8	12	207.8	3	⊛
noswot – o	478.3	146	290.1	61	3561.9
noswot – d	398.6	64	97.8	16	1835.5
pigeon-10 – o	661.1	23	61.23	10	1378.7
qiu – a	479.2	136	327.6	44	1149.3

⊛ – not solved in 12 hours

and TSA for the Bo160 and Bo320 instances, we studied the structure of these problems and their Pareto sets. It seems that for all instances from [11], the Pareto sets consist mostly of segments, all of which are relatively small and tightly packed along a curve in $\mathcal{OS}$ which has a seemingly parabolic shape. As discussed in [11], the TSA has the ability to employ LP based enhancements that allow the procedure to solve LPs in place of MILPs once a large number of Pareto solutions have been discovered. It seems that the case in which a Pareto sets consists mostly of relatively small segments is a situation in which this enhancement is quite powerful. Thus, we suspect that these enhancements play a significant role in the performace of TSA for the instances from [11], particularly the larger of these instances. We also point out that for the instances considered in Table 1, all the objective space algorithms are outperformed in terms of CPU time by the BB technique of [1].

From Table 2, recognize that SPEM outpeforms BB in terms of CPU time on all instances. This seems to be due to the more challenging structure and size of the instances considered here. Additinally, SPEM outpeforms TSA in terms of number of MILPs solved on all instances and in terms of CPU time on almost all instances. We believe that the results displayed in Table 2 are extremely important, even more so than those in Table 1, because they show that through the development of SPEM we have made progress towards developing a solver for BOMILP that can handle challenging instances, many of which have been developed from real-world problems.

7. Conclusion

We have introduced a new objective space search technique for solving BOMILP. We have argued the correctness and finiteness of our procedure and also conducted initial computational experiments. The results show that our procedure performs competitively with both the triangle splitting algorithm of Boland et al. [11] and the ϵ,Tabu-constraint method of Soylu and Yıldız [37] on small, relatively simple instances from the literature. The branch-and-bound algorithm of Adelgren and Gupte [1] clearly outperforms all of these methods, including the one presented here, for these small instances, though. On the other hand, initial results are promising for large, challenging instances from the literature, since our proposed method outperforms both the triangle splitting algorithm and the branch-and-bound method for almost all of these instances that were examined at the time of writing this paper.

In the near future we will perform more rigorous testing, utilizing all MIPlib instances from [1] and report these results. Additionally, we will seek to obtain code for the ϵ,Tabu-constraint method so that more extensive and more accurate results from its use can be reported. In the long term, we plan to modify the proposed procedure slightly so as to allow for parallelization, develop a parallelized version of the procedure, and report on its implementation and computational performance.

References

[1] N. Adelgren and A. Gupte. Branch-and-bound for biobjective mixed-integer programming. *Optimization Online*, 2016. URL `http://www.optimization-online.org/DB_FILE/2016/10/5676.pdf`.

[2] N. Adelgren, P. Belotti, and A. Gupte. Efficient storage of Pareto points in biobjective mixed integer programming. *Under Review*, 2014. URL `http://arxiv.org/abs/1411.6538`.

[3] Y. P. Aneja and K. P. Nair. Bicriteria transportation problem. *Management Science*, 25(1): 73–78, 1979.

[4] P. Armand. Finding all maximal efficient faces in multiobjective linear programming. *Mathematical Programming*, 61(1-3):357–375, 1993.

[5] P. Armand and C. Malivert. Determination of the efficient set in multiobjective linear programming. *Journal of Optimization Theory and Applications*, 70(3):467–489, 1991.

[6] P. Belotti, B. Soylu, and M. M. Wiecek. A branch-and-bound algorithm for biobjective mixed-integer programs. Technical report, Clemson University, December 2012. URL `http://www.clemson.edu/ces/math/technical_reports/belotti.bb-bicriteria.pdf`.

[7] H. Benson. Hybrid approach for solving multiple-objective linear programs in outcome space. *Journal of Optimization Theory and Applications*, 98(1):17–35, 1998.

[8] H. Benson and E. Sun. Outcome space partition of the weight set in multiobjective linear programming. *Journal of Optimization Theory and Applications*, 105(1):17–36, 2000.

[9] H. P. Benson. Further analysis of an outcome set-based algorithm for multiple-objective linear programming. *J. Optim. Theory Appl.*, 97(1):1–10, 1998. ISSN 0022-3239; 1573-2878/e. doi: 10.1023/A:1022614814789.

[10] H. P. Benson. An outer approximation algorithm for generating all efficient extreme points in the outcome set of a multiple objective linear programming problem. *Journal of Global Optimization*, 13(1):1–24, 1998.

[11] N. Boland, H. Charkhgard, and M. Savelsbergh. A criterion space search algorithm for biobjective mixed integer programming: The triangle splitting method. *INFORMS Journal on Computing*, 27(4):597–618, 2015.

[12] R. Borndörfer, S. Schenker, M. Skutella, and T. Strunk. Polyscip 2.0. URL `http://polyscip.zib.de/`.

[13] R. Borndörfer, S. Schenker, M. Skutella, and T. Strunk. Polyscip. In G.-M. Greuel, T. Koch, P. Paule, and A. Sommese, editors, *Mathematical Software ICMS 2016, 5th International Conference, Berlin, Germany, July 11-14, 2016, Proceedings*, volume 9725, pages 259 – 264, 2016. doi: 10.1007/978-3-319-42432-3_32.

[14] V. J. Bowman Jr. On the relationship of the tchebycheff norm and the efficient frontier of multiple-criteria objectives. In *Multiple criteria decision making*, pages 76–86. Springer, 1976.

[15] J. L. Cohon. *Multiobjective Programming and Planning*. New York: Academic Press, 2013.

[16] L. Csirmaz. Inner – a multiobjective linear program (molp) solver. URL `https://github.com/lcsirmaz/inner`.

[17] L. Csirmaz. Using multiobjective optimization to map the entropy region. *Computational Optimization and Applications*, 63(1):45–67, 2016.

[18] J. P. Dauer. Analysis of the objective space in multiple objective linear programming. *Journal of Mathematical Analysis and Applications*, 126(2):579–593, 1987.

[19] J. P. Dauer and Y.-H. Liu. Solving multiple objective linear programs in objective space. *European Journal of Operational Research*, 46(3):350–357, 1990.

[20] J. Ecker, N. S. Hegner, and I. Kouada. Generating all maximal efficient faces for multiple objective linear programs. *Journal of Optimization Theory and Applications*, 30(3):353–381, 1980.

[21] J. G. Ecker and I. Kouada. Finding all efficient extreme points for multiple objective linear programs. *Mathematical Programming*, 14(1):249–261, 1978.

[22] M. Ehrgott. *Multicriteria Optimization*, volume 2. Springer, 2005.

[23] M. Ehrgott and M. M. Wiecek. Multiobjective programming. In *Multiple criteria decision analysis: State of the art surveys*, pages 667–708. Springer, 2005.

[24] M. Ehrgott, A. Löhne, and L. Shao. A dual variant of bensons "outer approximation algorithm" for multiple objective linear programming. *Journal of Global Optimization*, 52(4):757–778, 2012.

[25] J. P. Evans and R. E. Steuer. A revised simplex method for linear multiple objective programs. *Mathematical Programming*, 5(1):54–72, 1973.

[26] A. M. Geoffrion. Proper efficiency and the theory of vector maximization. *Journal of Mathematical Analysis and Applications*, 22(3):618–630, 1968.

[27] A. H. Hamel, A. Löhne, and B. Rudloff. Benson type algorithms for linear vector optimization and applications. *Journal of Global Optimization*, 59(4):811–836, 2014.

[28] IBM-ILOG CPLEX 12.6 User's Manual. IBM, 2014.

[29] H. Isermann. The enumeration of the set of all efficient solutions for a linear multiple objective program. *Journal of the Operational Research Society*, 28(3):711–725, 1977.

[30] T. Koch, T. Achterberg, E. Andersen, O. Bastert, T. Berthold, R. E. Bixby, E. Danna, G. Gamrath, A. M. Gleixner, S. Heinz, A. Lodi, H. Mittelmann, T. Ralphs, D. Salvagnin, D. E. Steffy, and K. Wolter. MIPLIB 2010. *Mathematical Programming Computation*, 3(2):103–163, 2011.

[31] A. Löhne. *Vector optimization with infimum and supremum.* Springer Science & Business Media, 2011.

[32] A. Löhne and S. Schenker. Moplib. URL `http://moplib.zib.de/`.

[33] A. Löhne and B. Weißing. The vector linear program solver bensolve–notes on theoretical background. *European Journal of Operational Research*, 260(3):807–813, 2017.

[34] A. Löhne and B. Weißing. Bensolve: A free vlp solver, version 2.1. 2018. URL `http://www.bensolve.org/`.

[35] T. K. Ralphs, M. J. Saltzman, and M. M. Wiecek. An improved algorithm for solving biobjective integer programs. *Annals of Operations Research*, 147(1):43–70, 2006.

[36] B. Rudloff, F. Ulus, and R. Vanderbei. A parametric simplex algorithm for linear vector optimization problems. *Mathematical Programming*, 163(1-2):213–242, 2017.

[37] B. Soylu and G. B. Yıldız. An exact algorithm for biobjective mixed integer linear programming problems. *Computers & Operations Research*, 72:204–213, 2016.

[38] R. E. Steuer and E.-U. Choo. An interactive weighted tchebycheff procedure for multiple objective programming. *Mathematical programming*, 26(3):326–344, 1983.

[39] M. Zeleny. Linear multiobjective programming. *Lecture notes in economics and mathematical systems*, 95, 1974.

[40] S. Zionts and J. Wallenius. Identifying efficient vectors: some theory and computational results. *Operations Research*, 28(3-part-ii):785–793, 1980.